**PROJECT-10**

**EDGE DETECTION**

EE5356 Digital Image Processing

Dr. K. R. Rao

Submitted By:

Ashutosh Desai

UTA ID:1001388602

Email: - ashutoshrajesh.desai@mavs.uta.edu

SPRING-2018

# EE5356 LAB Assignment

# Edge detection:

Take a (256×256) or (512×512) 8-bit/pel image and perform the following edge detection operations:

1. Sobel Operator

2. Prewitt Operator

3. Robel operator

4. Laplacian of Gaussian

5. Canny’s Edge Detection

**Procedure:**

1) Read an image (any size up to 512x512).

2) Perform the edge detection using the techniques mentioned above.

3) Apply proper thresholding method and observe the difference in the image before and after applying thresholding.

4) Compare the final image obtained using default MATLAB Edge detection operator.

**References:**

**1) Anil K Jain, “Fundamentals of Digital Image Processing”, Prentice Hall Publication, pp 347-357**

**2)** **Rafael C. Gonzalez and Richard E.Woods, “Digital image**

**processing”, Prentice Hall Publication, Third Edition pp 706 - 723**

**EDGE DETECTION: -**

**Edge detection** is an image processing technique for finding the boundaries of objects within images. It works by **detecting** discontinuities in brightness.**Edge detection** is used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Edge detection includes a variety of mathematical methods that aim at identifying points in a [digital image](https://en.wikipedia.org/wiki/Digital_image) at which the [image brightness](https://en.wikipedia.org/wiki/Luminous_intensity) changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed *edges*. The same problem of finding discontinuities in one-dimensional signals is known as [step detection](https://en.wikipedia.org/wiki/Step_detection) and the problem of finding signal discontinuities over time is known as [change detection](https://en.wikipedia.org/wiki/Change_detection). Edge detection is a fundamental tool in [image processing](https://en.wikipedia.org/wiki/Image_processing), [machine vision](https://en.wikipedia.org/wiki/Machine_vision) and [computer vision](https://en.wikipedia.org/wiki/Computer_vision), particularly in the areas of [feature detection](https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)) and [feature extraction](https://en.wikipedia.org/wiki/Feature_extraction)

### **Types of edges**

Generally edges are of three types:

* Horizontal edges
* Vertical Edges
* Diagonal Edges

### **Types of Edge detection masks**

* Prewitt Operator
* Sobel Operator
* Robinson Compass Masks
* Krisch Compass Masks
* Laplacian Operator.

### **Prewitt Operator**

Prewitt operator is used for detecting edges horizontally and vertically.

### **Sobel Operator**

The sobel operator is very similar to Prewitt operator. It is also a derivate mask and is used for edge detection. It also calculates edges in both horizontal and vertical direction.

### **Robinson Compass Masks**

This operator is also known as direction mask. In this operator we take one mask and rotate it in all the 8 compass major directions to calculate edges of each direction.

### **Kirsch Compass Masks**

Kirsch Compass Mask is also a derivative mask which is used for finding edges. Kirsch mask is also used for calculating edges in all the directions.

### **Laplacian Operator**

Laplacian Operator is also a derivative operator which is used to find edges in an image. Laplacian is a second order derivative mask. It can be further divided into positive laplacian and negative laplacian.

All these masks find edges. Some find horizontally and vertically, some find in one direction only and some find in all the directions. The next concept that comes after this is sharpening which can be done once the edges are extracted from the image

* **APPROACHES**

There are many methods for edge detection, but most of them can be grouped into two categories, search-based and [zero-crossing](https://en.wikipedia.org/wiki/Zero_crossing) based. The search-based methods detect edges by first computing a measure of edge strength, usually a [first-order derivative expression](https://en.wikipedia.org/wiki/First-order_derivative_expression) such as the gradient magnitude, and then searching for local directional maxima of the gradient magnitude using a computed estimate of the local orientation of the edge, usually the gradient direction. The zero-crossing based methods search for zero crossings in a [second-order derivative expression](https://en.wikipedia.org/wiki/Second-order_derivative_expression) computed from the image in order to find edges, usually the zero-crossings of the [Laplacian](https://en.wikipedia.org/wiki/Laplacian) or the zero-crossings of a non-linear differential expression. As a pre-processing step to edge detection, a smoothing stage, typically [Gaussian](https://en.wikipedia.org/wiki/Gaussian" \o "Gaussian)smoothing, is almost always applied (see also [noise reduction](https://en.wikipedia.org/wiki/Noise_reduction)).

The edge detection methods that have been published mainly differ in the types of smoothing filters that are applied and the way the measures of edge strength are computed. As many edge detection methods rely on the computation of image gradients, they also differ in the types of filters used for computing gradient estimates in the x- and y-directions.

A survey of a number of different edge detection methods can be found in (Ziou and Tabbone 1998);[[6]](https://en.wikipedia.org/wiki/Edge_detection#cite_note-6)see also the encyclopedia articles on edge detection in Encyclopedia of Mathematics[[3]](https://en.wikipedia.org/wiki/Edge_detection#cite_note-lin95-3) and Encyclopedia of Computer Science and Engineering

1. **Canny**

[John Canny](https://en.wikipedia.org/wiki/John_Canny) considered the mathematical problem of deriving an optimal smoothing filter given the criteria of detection, localization and minimizing multiple responses to a single edge.[[8]](https://en.wikipedia.org/wiki/Edge_detection#cite_note-8) He showed that the optimal filter given these assumptions is a sum of four exponential terms. He also showed that this filter can be well approximated by first-order derivatives of Gaussians. Canny also introduced the notion of non-maximum suppression, which means that given the presmoothing filters, edge points are defined as points where the gradient magnitude assumes a local maximum in the gradient direction. Looking for the zero crossing of the 2nd derivative along the gradient direction was first proposed by [Haralick](https://en.wikipedia.org/wiki/Haralick" \o "Haralick).[[9]](https://en.wikipedia.org/wiki/Edge_detection#cite_note-9) It took less than two decades to find a modern geometric variational meaning for that operator that links it to the [Marr–Hildreth](https://en.wikipedia.org/wiki/Marr%E2%80%93Hildreth_algorithm) (zero crossing of the Laplacian) edge detector. That observation was presented by [Ron Kimmel](https://en.wikipedia.org/wiki/Ron_Kimmel) and [Alfred Bruckstein](https://en.wikipedia.org/w/index.php?title=Alfred_Bruckstein&action=edit&redlink=1).[[10]](https://en.wikipedia.org/wiki/Edge_detection#cite_note-10)

Although his work was done in the early days of computer vision, the [Canny edge detector](https://en.wikipedia.org/wiki/Canny_edge_detector) (including its variations) is still a state-of-the-art edge detector.[[11]](https://en.wikipedia.org/wiki/Edge_detection#cite_note-11) Edge detectors that perform better than the Canny usually require longer computation times or a greater number of parameters.

The Canny–Deriche detector was derived from similar mathematical criteria as the Canny edge detector, although starting from a discrete viewpoint and then leading to a set of recursive filters for image smoothing instead of [exponential filters](https://en.wikipedia.org/w/index.php?title=Exponential_filter&action=edit&redlink=1) or Gaussian filters.[[12]](https://en.wikipedia.org/wiki/Edge_detection#cite_note-12)

The [differential edge detector](https://en.wikipedia.org/wiki/Edge_detection#Differential_edge_detection) described below can be seen as a reformulation of Canny's method from the viewpoint of differential invariants computed from a [scale space representation](https://en.wikipedia.org/wiki/Scale_space_representation) leading to a number of advantages in terms of both theoretical analysis and sub-pixel implementation. In that aspect, [Log Gabor filter](https://en.wikipedia.org/wiki/Log_Gabor_filter) have been shown to be a good choice to extract boundaries in natural scenes

### **Thresholding and linking**

Once we have computed a measure of edge strength (typically the gradient magnitude), the next stage is to apply a threshold, to decide whether edges are present or not at an image point. The lower the threshold, the more edges will be detected, and the result will be increasingly susceptible to [noise](https://en.wikipedia.org/wiki/Image_noise) and detecting edges of irrelevant features in the image. Conversely a high threshold may miss subtle edges, or result in fragmented edges.

If the edge is applied to just the gradient magnitude image, the resulting edges will in general be thick and some type of edge thinning post-processing is necessary. For edges detected with non-maximum suppression however, the edge curves are thin by definition and the edge pixels can be linked into edge polygon by an edge linking (edge tracking) procedure. On a discrete grid, the non-maximum suppression stage can be implemented by estimating the gradient direction using first-order derivatives, then rounding off the gradient direction to multiples of 45 degrees, and finally comparing the values of the gradient magnitude in the estimated gradient direction.

A commonly used approach to handle the problem of appropriate thresholds for thresholding is by using [thresholding](https://en.wikipedia.org/wiki/Adaptive_thresholding) with [hysteresis](https://en.wikipedia.org/wiki/Hysteresis). This method uses multiple thresholds to find edges. We begin by using the upper threshold to find the start of an edge. Once we have a start point, we then trace the path of the edge through the image pixel by pixel, marking an edge whenever we are above the lower threshold. We stop marking our edge only when the value falls below our lower threshold. This approach makes the assumption that edges are likely to be in continuous curves, and allows us to follow a faint section of an edge we have previously seen, without meaning that every noisy pixel in the image is marked down as an edge. Still, however, we have the problem of choosing appropriate thresholding parameters, and suitable thresholding values may vary over the image.

### **Edge thinning**

### **Edge thinning is a technique used to remove the unwanted spurious points on the edges in an image. This technique is employed after the image has been filtered for noise (using median, Gaussian filter etc.), the edge operator has been applied (like the ones described above) to detect the edges and after the edges have been smoothed using an appropriate threshold value. This removes all the unwanted points and if applied carefully, results in one pixel thick edge elements.**

**Figure: - Edge Detection**

**MATLAB SCRIPT:**

function[]=EDGE()

clc;

clear all;

close all;

Image=imread('D:\STUDY\DIP\Test img\girl512.bmp');

Image=double(Image);

%Sobel Method

A1 = [1,0,-1;2,0,-2;1,0,-1];

B1 = A1'';

G\_Sobel = conv2(Image,A1,'same');

H\_Sobel = conv2(Image,B1,'same');

Sobel\_M = G\_Sobel+H\_Sobel;

a = mean(mean(Image));

[Threshold\_Sobel] = edge\_fn(Sobel\_M,a);

subplot(2,2,4);imshow(uint8(Threshold\_Sobel));title('Thresholded Edge');

subplot(2,2,1);imshow(uint8(G\_Sobel));title('Edge along X');

subplot(2,2,2);imshow(uint8(H\_Sobel));title('Edge along Y');

subplot(2,2,3);imshow(uint8(Sobel\_M));title('Edges for Sobel method');

%Prewitt Method

X2 = [1,0,-1;1,0,-1;1,0,-1];

Y2 = X2';

G\_Prewitt = conv2(Image,X2,'same');

H\_Prewitt = conv2(Image,Y2,'same');

Prewitt\_M = G\_Prewitt+H\_Prewitt;

[Threshold\_Prewitt]=edge\_fn(Prewitt\_M,a);

figure;

subplot(2,2,4);imshow(uint8(Threshold\_Prewitt));title('Thresholded Edge');

subplot(2,2,1);imshow(uint8(G\_Prewitt));title('Edges along X');

subplot(2,2,2);imshow(uint8(H\_Prewitt));title('Edges along Y');

subplot(2,2,3);imshow(uint8(Prewitt\_M));title('Edge Detection by Prewitt method');

%Roberts Method

X3 = [1,0;0,-1];

Y3 = [0,1;-1,0];

G\_Roberts = conv2(Image,X3,'same');

H\_Roberts = conv2(Image,Y3,'same');

Roberts\_M = G\_Roberts+H\_Roberts;

[Threshold\_Roberts] = edge\_fn(Roberts\_M,30);

figure;

subplot(2,2,4);imshow(uint8(Threshold\_Roberts));title('Thresholded Edge');

subplot(2,2,1);imshow(uint8(G\_Roberts));title('Edge using mask 1');

subplot(2,2,2);imshow(uint8(H\_Roberts));title('Edge using mask 2');

subplot(2,2,3);imshow(uint8(Roberts\_M));title('Edge Detection by Roberts method');

%Gaussian Method

Gau = [2,4,5,4,2;4,9,12,9,4;5,12,15,12,5;4,9,12,9,4;2,4,5,4,2];

Gau = Gau./159;

H2 = [1,1,1;1,-8,1;1,1,1];

Gau\_f = conv2(Image,Gau,'same');

Gau2\_f = conv2(Gau\_f,H2,'same');

[Threshold\_Gaussian] = edge\_fn(Gau2\_f,20);

figure;

subplot(2,2,1);imshow(uint8(Image));title('Original Image for Gaussian method');

subplot(2,2,2);imshow(uint8(Gau\_f));title('Gaussian Filtered Image');

subplot(2,2,3);imshow(uint8(Gau2\_f));title('Edge Detection');

subplot(2,2,4);imshow(uint8(Threshold\_Gaussian));title('Thresholded Edge');

%Canny Edge

im\_can = conv2(Image,Gau,'same');

X\_Can = [-1,0,1;-2,0,2;-1,0,1];

Y\_Can = [1,2,1;0,0,0;-1,-2,-1];

G\_Can = conv2(im\_can,X\_Can,'same');

H\_Can = conv2(im\_can,Y\_Can,'same');

Canny\_M = G\_Can+H\_Can;

[Threshold\_Canny] = edge\_fn(Canny\_M,50);

figure;

subplot(3,2,1);imshow(uint8(Image));title('Original Image');

subplot(3,2,3);imshow(uint8(X\_Can));title('Edge along X');

subplot(3,2,5);imshow(uint8(Canny\_M));title('Overall Edge Detection');

subplot(3,2,2);imshow(uint8(im\_can));title('Filtered Image');

subplot(3,2,4);imshow(uint8(Y\_Can));title('Edge along Y');

subplot(3,2,6);imshow(uint8(Threshold\_Canny));title('Thresholded Edge');

end

%Function:

function [out]=edge\_fn(input,K)

for i=1:1:512

for j=1:1:512

if input(i,j)>K

out(i,j)=255;

else

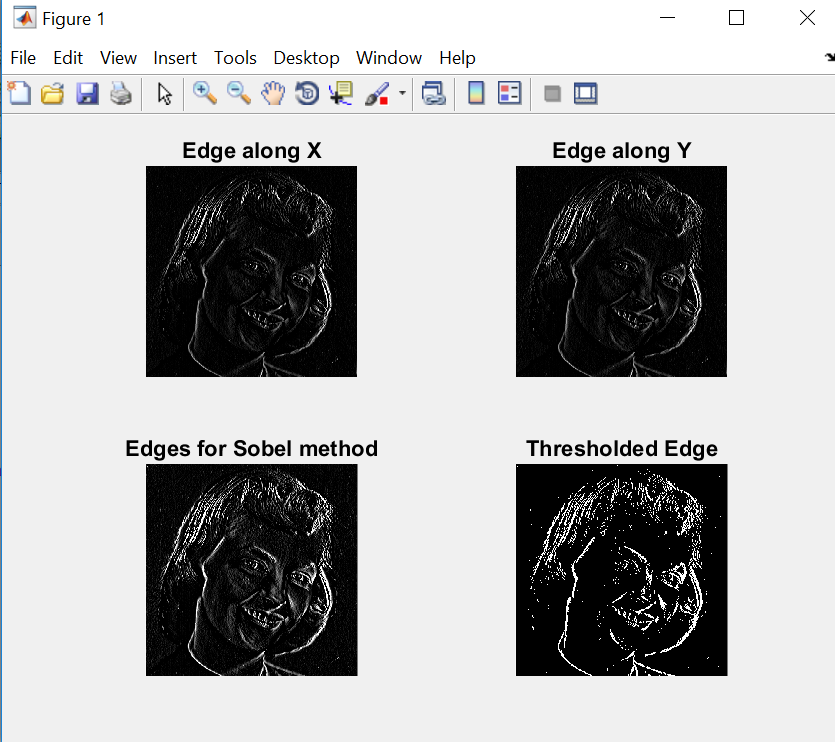
out(i,j)=0;

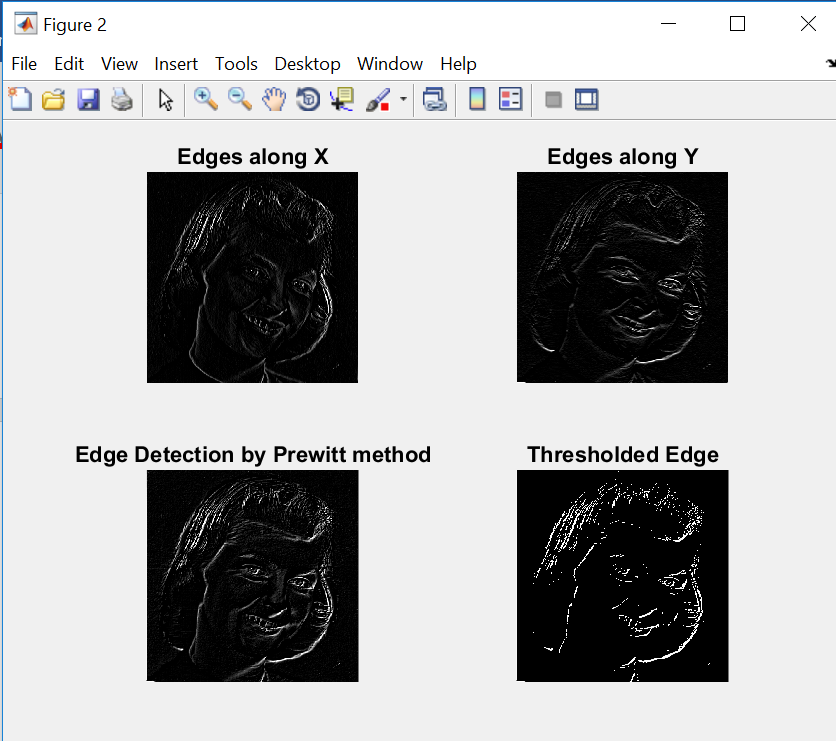
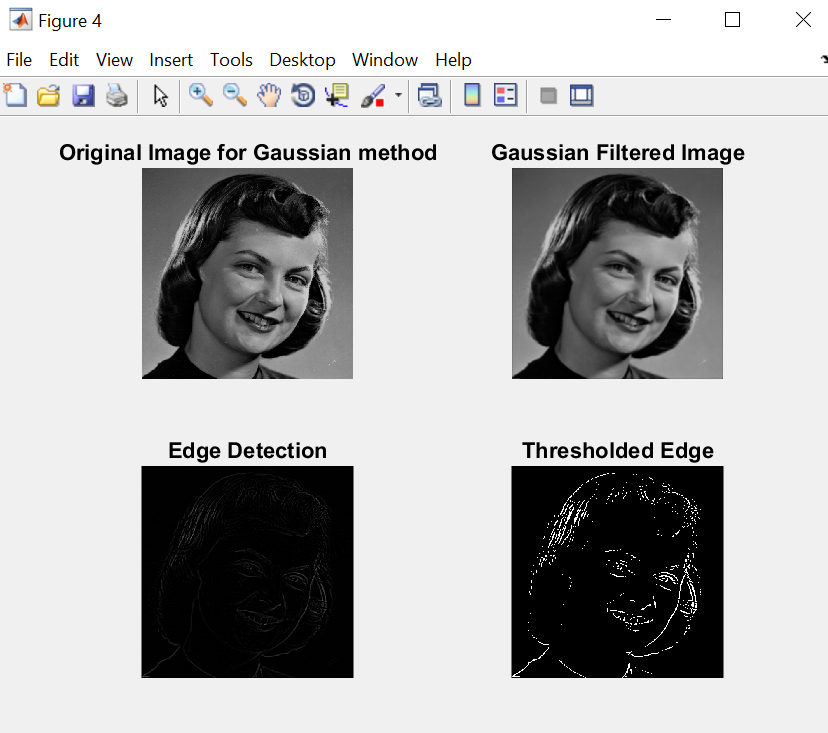
end

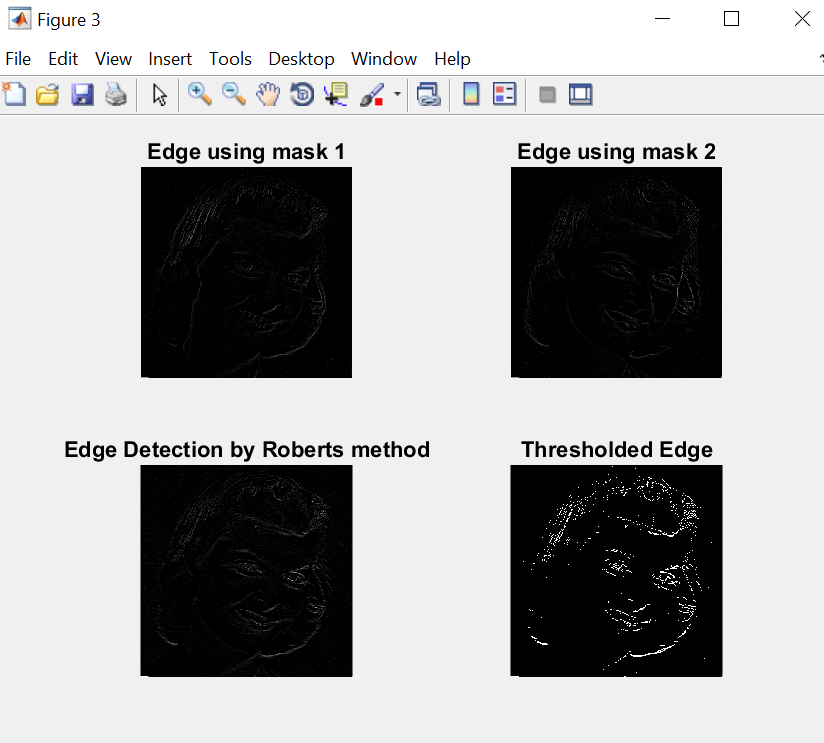
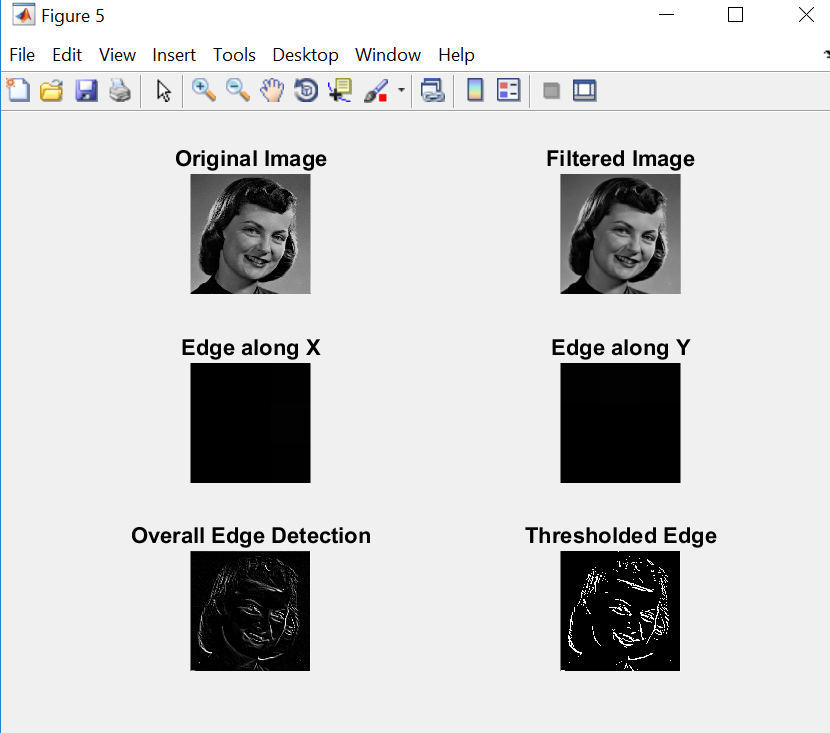
end

end

en

**OUTPUT: -**





**OPERATION:**

* Edge detection is a terminology in image processing and computer vision, particularly in the areas of feature detection and feature extraction, to refer to algorithms which aim at identifying points in a digital image at which the image brightness changes sharply or more formally has discontinuities.
* The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges.
* The Prewitt operator uses two 3×3 kernels which are [convolved](http://en.wikipedia.org/wiki/Convolution) with the original image to calculate approximations of the derivatives - one for horizontal changes, and one for vertical.
* The http://homepages.inf.ed.ac.uk/rbf/HIPR2/mote.gifRoberts Cross operator performs a simple, quick to compute, 2-D spatial gradient measurement on an image.
* The canny edge detector first smoothes the image to eliminate and noise. It then finds the image gradient to highlight regions with high spatial derivatives.

**CONCLUSION:-**  
 Edge detection is an image processing technique for finding the boundaries of objects within images. It works by detecting discontinuities in brightness. Edge detections used for image segmentation and data extraction in areas such as image processing, computer vision, and machine vision.

Thus, in this project, we have studied about the various edge detection techniques and observed the variations in the output image when its passed through that detector.